ELSEVIER

Theoretical Computer Science 259 (2001) 513–531

Theoretical Computer Science

www.elsevier.com/locate/tcs

Signed genome rearrangement by reversals and transpositions: models and approximations ☆

Guo-Hui Lin *, Guoliang Xue

Department of Computer Science, The University of Vermont, Burlington, VT 05405, USA

Received June 1999; revised October 1999
Communicated by D.-Z. Du

Abstract

An important problem in computational biology is the genome rearrangement using reversals and transpositions. Analysis of genome evolving by reversals and transpositions leads to a combinatorial optimization problem of sorting by reversals and transpositions, i.e., sorting of a permutation using reversals and transpositions of arbitrary fragments. The reversal operation works on a single segment of the genome by reversing the selected segment. Two kinds of transpositions have been studied in the literature. The first kind of transposition operations delete a segment of the genome and insert it into another position in the genome. The second kind of transposition operations delete a segment of the genome and insert its inverse into another position in the genome. Both transposition operations can be viewed as operations working on two consecutive segments. In this paper, we introduce a third transposition operation which works on two consecutive segments and study sorting of a signed permutation by reversals and transpositions. By allowing reversals and the first kind of transpositions, or reversals and the first two kinds of transpositions, or reversals and all three kinds of transpositions, we have three problem models. After establishing a common lower bound on the number of operations needed, we present a unified 2-approximation algorithm for all these problems. Finally, we present a better 1.75-approximation for the third problem. © 2001 Published by Elsevier Science B.V.

Keywords: Approximation algorithms; Genome rearrangement; Sorting of permutations

1. Introduction

Traditionally, the evolutionary distance between two species had been measured using the *edit distance* between two DNA sequences, i.e., the minimum number of

☆ This research was supported in part by the Army Research Office grant DAAH04-96-10233 and by the National Science Foundation grants ASC-9409285 and OSR-9350540. An extended abstract of a preliminary version of this paper has appeared in 99.

* Corresponding author. Fax: +802-656-0696.
E-mail addresses: ghlin@cs.uvm.edu (G.-H. Lin), xue@cs.uvm.edu (G. Xue).

0304-3975/01/$ - see front matter © 2001 Published by Elsevier Science B.V.
PII: S0304-3975(00)00038-4

insertions, deletions and substitutions required to change one DNA sequence to another. Note that each of the three operations either inserts a single character, or deletes a single character, or replaces one character with another. People had believed that this is the way evolutionary changes take place, until the late 1980s when J. Palmer and his colleagues discovered a remarkable pattern of evolutionary change in plant organelles. They compared the mitochondrial genomes of *Brassica oleracea* (cabbage) and *Brassica campestris* (turnip), which are very closely related (many genes are 99–99.9% identical). To their surprise, these molecules, which are almost identical in gene sequence, differ dramatically in gene order. This discovery and many other studies in the last decade convincingly proved that genome rearrangement is a common model of molecular evolution in mitochondrial, chloroplast, viral and bacterial DNA [1]. That is, at the chromosome level, genetic sequences mutate by many global operations such as the inversion of a substring (*reversal*), the deletion and subsequent re-insertion of a substring far from its original position (*transposition*), duplication of a substring and the exchange of prefixes or suffixes of two sequences in the same organism (*translocation*). Here we are interested in reversals and transpositions.

The study of genome rearrangement by reversals and transpositions involves solving a combinatorial puzzle to find a shortest series of reversals and transpositions to transform one genome into another. This (minimum) number of operations required is taken as the *evolutionary distance* between these two genomes. For genomes consisting of a small number of "conserved blocks", the most parsimonious scenarios may be found in a brute-force style. However, for genomes consisting of a large number of blocks, finding the optimal solution becomes difficult. Previously, much research focused on genome rearrangement by reversals only, which has been shown to be NP-hard [4]. In the design of approximation algorithms, Kececioglu and Sankoff [7] presented a 2-approximation. This performance ratio of 2 was further improved to 1.75 by Bafna and Pevzner [2]. The problem of sorting by transpositions only is also of interest. Bafna and Pevzner [3] designed a 1.5-approximation for this problem, although whose computational complexity is still unknown.

Any operation applied on a genome should not break the conserved blocks. From this point of view, we may represent a genome with n conserved blocks by a permutation π on set $\{1,2,\ldots,n\}$, where each element π_j in π represents a conserved block. In this way, it is easy to define reversals and transpositions performed on permutations which correspond to the reversals and transpositions performed on genomes; and thus a series of reversals and transpositions transforming one permutation π into another σ maps a series of reversals and transpositions transforming one genome into another, correspondingly. We will study the latter instead of the former for the sake of simplicity of exposition.

Let $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$ be a permutation of $\{1,2,\ldots,n\}$. For $1\leqslant i<j\leqslant n+1$, a *reversal* $r(i,j)$ is the permutation

$$\begin{pmatrix} 1,\ldots,i-1,\boxed{\mathbf{i},\mathbf{i+1},\ldots,\mathbf{j-1}},j,\ldots,n \\ 1,\ldots,i-1,\boxed{\mathbf{j-1},\ldots,\mathbf{i+1},\mathbf{i}},j,\ldots,n \end{pmatrix}.$$

Therefore, $\pi \cdot r(i,j) = (\pi_1, \ldots, \pi_{i-1}, \pi_{j-1}, \ldots, \pi_{i+1}, \pi_i, \pi_j, \ldots, \pi_n)$, i.e., $\pi \cdot r(i,j)$ has the effect of inverting the order of $\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}$. For $1 \leqslant i < j \leqslant n+1$ and $1 \leqslant k \leqslant n+1$ with $k \notin [i,j]$, a *transposition* $t(i,j,k)$ is the permutation

$$\begin{pmatrix} 1,\ldots,i-1,\boxed{\mathbf{i},\mathbf{i+1},\ldots,\mathbf{j-1}},\boxed{\mathbf{j},\ldots,\mathbf{k-1}},k,\ldots,n \\ 1,\ldots,i-1,\boxed{\mathbf{j},\ldots,\mathbf{k-1}},\boxed{\mathbf{i},\mathbf{i+1},\ldots,\mathbf{j-1}},k,\ldots,n \end{pmatrix}.$$

That is, $\pi \cdot t(i,j,k) = (\pi_1, \ldots, \pi_{i-1}, \pi_j, \ldots, \pi_{k-1}, \pi_i, \ldots, \pi_{j-1}, \ldots, \pi_k, \ldots, \pi_n)$. In other words, $\pi \cdot t(i,j,k)$ has the effect of moving $\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}$ to a new location of π before π_k.

Given permutations π and σ, the *rearrangement distance problem* is to find a series of reversals and transpositions $\rho_1, \rho_2, \ldots, \rho_d$ such that $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_d = \sigma$ and d is minimum. We call d the *rearrangement distance* between π and σ. Since the rearrangement distance between π and σ equals to the rearrangement distance between $\sigma^{-1}\pi$ and the *identity* permutation $\mathscr{I} = (1,2,\ldots,n)$, sorting π is the problem of finding the rearrangement distance, denoted by $d(\pi)$, between π and $\mathscr{I}$.

Now we can define an *α-approximation algorithm*. An α-approximation algorithm for sorting a permutation is an algorithm which, given any π, finds a series of operations $\rho_1, \ldots, \rho_d$ such that $\rho_1, \ldots, \rho_d$ sort π into $\mathscr{I}$ and d satisfies $d(\pi) \leqslant d \leqslant \alpha d(\pi)$.

A signed permutation is a permutation π on $\{1,2,\ldots,n\}$ with $+$ or $-$ sign associated with every element π_j of π. For example, $(-1,-4,+3,+2)$ is a signed permutation of $\{1,2,3,4\}$. The identity of signed permutation is $(+1,+2,\ldots,+n)$. It is noted that signed permutations are more relevant to genome rearrangement, since genes are oriented in DNA sequences. For sorting signed genomes by reversals only, Bafna and Pevzner [2] designed a 1.5-approximation, and later Hannenhalli and Pevzner [6] presented a polynomial time algorithm which finds the minimum number of reversals for a signed permutation. This is the first polynomial algorithm for a realistic model of genome rearrangement problems.

For a signed permutation, a reversal on a block not only inverses the order of the elements in the block, but also changes the signs. It is clear that by transpositions only, sorting any signed permutation into the identity is impossible. Nonetheless, transpositions do help save some number of operations, if they are used together with reversals. For example, sorting the (signed) permutation $(-1,-4,+3,+2)$ in Fig. 1 requires exactly four reversals, according to the algorithm by Hannenhalli and Pevzner [6]; while one reversal and two transpositions, if allowed, suffice to sort it into identity $\mathscr{I} = (+1,+2,+3,+4)$.

In a recent paper [5], Gu et al. proposed a new kind of transpositions $rt(i,j,k)$, defined as the permutation (for $1 \leqslant i < j \leqslant n+1$ and $1 \leqslant k \leqslant n+1$ with $k \notin [i,j]$)

$$\begin{pmatrix} 1,\ldots,i-1,\boxed{\mathbf{i},\mathbf{i+1},\ldots,\mathbf{j-1}},\boxed{\mathbf{j},\ldots,\mathbf{k-1}},k,\ldots,n \\ 1,\ldots,i-1,\boxed{\mathbf{j},\ldots,\mathbf{k-1}},\boxed{\mathbf{j-1},\ldots,\mathbf{i+1},\mathbf{i}},k,\ldots,n \end{pmatrix}.$$

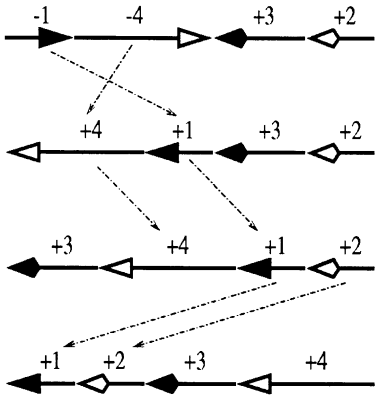


Fig. 1. Sorting $(-1,-4,+3,+2)$ into identity.

Therefore, $\pi \cdot rt(i,j,k) = (\pi_1,\ldots,\pi_{i-1},\pi_j,\ldots,\pi_{k-1},\pi_{j-1},\ldots,\pi_i,\ldots,\pi_k,\ldots,\pi_n)$, i.e., $\pi \cdot rt(i,j,k)$ has the effect of inverting $\pi_i,\pi_{i+1},\ldots,\pi_{j-1}$ and moving it to a new location of π before π_k. They then designed a 2-approximation algorithm for sorting signed permutations using these three kinds of operations, which runs in time $O(n^2)$ (n is the number of elements in the permutation).

In this paper, we propose a new kind of transpositions $rr(i,j,k)$ and consider three versions of sorting signed permutations: (1) by reversals and transpositions $t(i,j,k)$; (2) by reversals and transpositions $t(i,j,k)$ and $rt(i,j,k)$; (3) by reversals and transpositions $t(i,j,k)$, $rt(i,j,k)$ and $rr(i,j,k)$. (The second version is the problem considered in [5].) After establishing a common lower bound on the number of operations needed, we present a unified approximation algorithm with performance guarantee of 2 for these three versions. In the next section, we present some definitions and conventions needed throughout this paper. The common lower bound and the unified 2-approximation algorithm are presented in Section 3. Section 4 describes a better approximation algorithm for the third version, whose performance ratio is 1.75. We conclude the paper in Section 5.

2. Preliminaries

For an unsigned permutation $\pi = (\pi_1,\pi_2,\ldots,\pi_n)$, (π_i,π_{i+1}) is called a *breakpoint* if $|\pi_i - \pi_{i+1}| \neq 1$. As there is no breakpoint in the identity permutation $\mathcal{I}$, the sorting problem is equivalent to eliminating all breakpoints in π, using as few operations as possible.

Note that operation $r(i,j)$ acts on a single segment of elements, while transpositions $t(i,j,k)$ and $rt(i,j,k)$ can be viewed as acting on two consecutive segments of elements. If we take two consecutive segments and put them back with possible inversions, there are exactly four possible operations. $t(i,j,k)$ and $rt(i,j,k)$ are two of the four

operations. One of the other two reduces to reversal. The only operation which has not been studied in the literature is defined in the following, which we call the third kind of transpositions $rr(i,j,k)$. A transposition $rr(i,j,k)$ is the permutation (for $1 \leqslant i < j \leqslant n+1$ and $1 \leqslant k \leqslant n+1$ with $k \notin [i,j]$)

$$\begin{pmatrix} 1,\ldots,i-1,\boxed{\mathbf{i},\mathbf{i+1},\ldots,\mathbf{j-1}},\boxed{\mathbf{j},\mathbf{j+1},\ldots,\mathbf{k-1}},k,\ldots,n \\ 1,\ldots,i-1,\boxed{\mathbf{j-1},\ldots,\mathbf{i+1},\mathbf{i}},\boxed{\mathbf{k-1},\ldots,\mathbf{j+1},\mathbf{j}},k,\ldots,n \end{pmatrix}.$$

Therefore, $\pi \cdot rr(i,j,k) = (\pi_1,\ldots,\pi_{i-1},\pi_{j-1},\ldots,\pi_i,\pi_{k-1},\ldots,\pi_j,\ldots,\pi_k,\ldots,\pi_n)$, i.e., $\pi \cdot rr(i,j,k)$ has the effect of inverting the segments $(\pi_i,\pi_{i+1},\ldots,\pi_{j-1})$ and $(\pi_j,\pi_{j+1},\ldots,\pi_{k-1})$ at the same time and then putting them back to the permutation. Although we are not aware of any biological evolutions via the third kind of transpositions, we believe that the study of genome arrangement using such operations is useful, as well as the second kind of transpositions. After all, global operations such as reversals and the first kind of transpositions were not observed until the late 1980s.

In the case where π is a signed permutation of n elements, define its unsigned *image permutation* π^* of $2n$ elements as follows: replace $+i$ with $(2i-1,2i)$ and replace $-i$ with $(2i,2i-1)$. In this way, the signed identity $\mathscr{I}$ has its image permutation $\mathscr{I}^* = (1,2,3,4,\ldots,2n-1,2n)$. In what follows, we assume that any operation on the image permutation π^* of a signed permutation π never breaks any pair of $(2i-1,2i)$ or $(2i,2i-1)$ which correspond to $+i$ or $-i$ in the corresponding signed permutation. We will see later that this assumption is not restrictive by Lemma 3.1. It follows that any sequence of operations transforming π into $\mathscr{I}$ one-to-one corresponds to a sequence of operations transforming π^* into $\mathscr{I}^*$. In other words, the signed permutation π and its image (unsigned) permutation π^* are equivalent for our purpose. In the rest of this paper, by permutation we mean the image permutation for a signed permutation, unless otherwise specified. It is clear that an operation $t(i,j,k)$ $(r(i,j), rt(i,j,k), rr(i,j,k))$ performed on the signed permutation π corresponds to an operation $t^*(2i-1,2j-1,2k-1)$ performed on the image permutation π^*. Therefore, in all later-mentioned operations, the parameters i,j,k involved are all odd integers.

If (π_{i-1},π_i) is a breakpoint, we say operation $r(i,j)$ $(t(i,j,k), rt(i,j,k), rr(i,j,k)$, respectively) acts on breakpoint (π_{i-1},π_i). Similarly, if (π_{i-1},π_i), (π_{j-1},π_j) and (π_{k-1},π_k) are breakpoints, we say operation $t(i,j,k)$ ($rt(i,j,k)$, $rr(i,j,k)$, respectively) acts on breakpoints (π_{i-1},π_i), (π_{j-1},π_j) and (π_{k-1},π_k).

Note that every reversal and every one of the three kinds of transpositions acts on at most three breakpoints, and thus it can eliminate at most three breakpoints. Therefore $b(\pi)/3$ is a trivial lower bound for all three aforementioned sorting problems, where $b(\pi)$ is the number of breakpoints in π. In the following, we will derive a better lower bound using a more careful analysis. Let us first define the *breakpoint graph* of a permutation [6].

Let $\pi = (\pi_1,\pi_2,\ldots,\pi_{2n})$ be an arbitrary permutation. Extend π by adding $\pi_0 = 0$ and $\pi_{2n+1} = 2n+1$. Let $i \sim j$ if $|i-j| = 1$. A pair of consecutive elements (π_i,π_{i+1}) is an *adjacency* if $\pi_i \sim \pi_{i+1}$ (otherwise, a breakpoint). The *breakpoint graph* of π is an

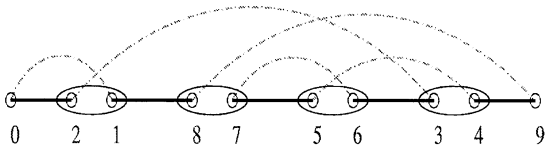


Fig. 2. $G(\pi)$: $\pi = (-1, -4, +3, +2)$.

edge-colored graph $G(\pi)$:

- There are $2n+2$ nodes $0, 1, 2, \ldots, 2n-1, 2n, 2n+1$ in $G(\pi)$;
- There is a *gray edge* between i and j if $i \sim j$ and i and j are not consecutive in π;
- There is a *black edge* between i and j if (i, j) is a breakpoint.

The graph $G(\pi)$ of $\pi = (-1, -4, +3, +2)$ is given in Fig. 2.

A sequence of distinct nodes $v_1, v_2, \ldots, v_m$ is called a *segment* in $G(\pi)$ if $(v_i, v_{i+1}) \in E(G)$, $1 \leqslant i \leqslant m-1$. A sequence of nodes $v_1, v_2, \ldots, v_m = v_1$ is called a *cycle* in $G(\pi)$ if $(v_i, v_{i+1}) \in E(G)$, for $1 \leqslant i \leqslant m-1$. A segment/cycle is *alternating* if the colors of edges in the segment/cycle alternate. Define the length of an alternating cycle the number of black edges (breakpoints) in the cycle. The following lemma lists some simple facts which are needed in the paper. The proof of the lemma is trivial and is omitted.

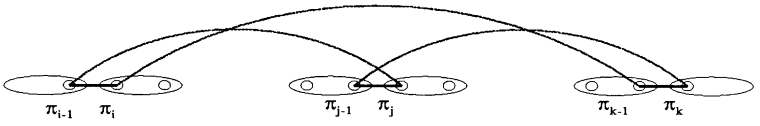
Lemma 2.1. *For the breakpoint graph $G(\pi)$ of a permutation π,*

1. *the gray degree and black degree of each node are the same and equal to either 0 or 1;*
2. *each connected component of $G(\pi)$ is an alternating cycle;*
3. *each alternating cycle has at least two gray (black) edges;*
4. *there is no edge in $G(\mathcal{I})$, the breakpoint graph of the identity permutation $\mathcal{I}$.*

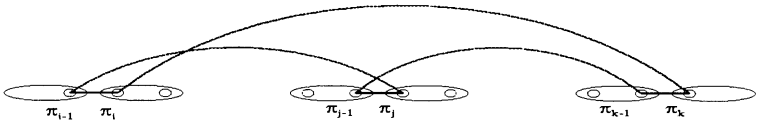
In what follows, by cycle we mean an alternating cycle. Call a cycle a k-cycle if its length is k. A k-cycle is an odd cycle if k is odd, otherwise it is called an even cycle. Let $o(\pi)$ be the number of odd cycles in $G(\pi)$.

3. A Unified 2-Approximation Algorithm

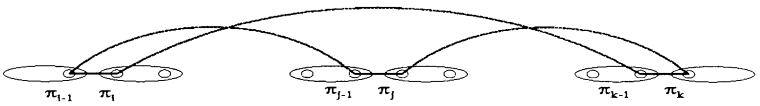
Let ρ be an operation, and $\pi' = \pi \cdot \rho$. Let $b(\pi')$ be the number of breakpoints in π'. Note that ρ acts on at most three breakpoints, $|b(\pi) - b(\pi')| \leqslant 3$. This implies $d(\pi) \geqslant b(\pi)/3$. In order to get a better lower bound of $d(\pi)$, we notice that only transpositions act on three breakpoints. Moreover, a transposition $\rho = t(i, j, k)$ ($rt(i, j, k)$, $rr(i, j, k)$) reduces the number of breakpoints by three if and only if (π_{i-1}, π_i), (π_{j-1}, π_j) and (π_{k-1}, π_k) are breakpoints belonging to a 3-cycle, and this cycle has a configuration isomorphic to that shown in Fig. 3 (a), (b), (c), respectively). From this, to eliminate a cycle of length k, we need at least $k/2$ operations for k even and at least $(k-1)/2$ operations for k odd. This suggests that $d(\pi)$ is at least $(b(\pi) - o(\pi))/2$, where $o(\pi)$ is



(a) by $t(i,j,k)$.



(b) by $rt(i,j,k)$.



(c) by $rr(i,j,k)$.

Fig. 3. The cycles where a transposition can eliminate three breakpoints.

the number of odd cycles in $G(\pi)$. We show this in the following.

Lemma 3.1. *Any sequence of operations transforming permutation π into $\mathscr{I}$ can be regulated in such a way that each of the operations acts on breakpoints only.*

Proof. Suppose that $\rho_1, \rho_2, \ldots, \rho_k$ is a sequence of operations transforming permutation π into identity $\mathscr{I}$. We may first regulate ρ_k to acts on breakpoints only. And then regulate ρ_{k-1} (and changing ρ_k accordingly) to acts on breakpoints only. Repeatedly, applying this process in the reversed order of operations, we will finally get another sequence of operations, which act on breakpoints only, transforming permutation π into identity $\mathscr{I}$. □

From the above lemma, we need only to consider those operations acting on breakpoints only.

Lemma 3.2. *For a permutation π and an operation ρ with $\pi' = \pi \cdot \rho$, $\Delta(\rho) = b(\pi) - o(\pi) - b(\pi') + o(\pi') = -2, 0$ or 2, which is the* decrement *of operation ρ.*

Proof. Note first that any operation reduces the parameter $b(\pi)$ by at most three, and whenever it reduces three the corresponding three breakpoints must belong to a same cycle and this cycle has only three black edges. Therefore, this operation reduces the

parameter $b(\pi) - o(\pi)$ by at most two, i.e., $\Delta(\rho) = b(\pi) - o(\pi) - b(\pi') + o(\pi') \geqslant -2$. Now the parity consideration implies $\Delta(\rho) = -2, 0$ or 2. □

Theorem 3.1. *For any permutation* π, $d(\pi) \geqslant (b(\pi) - o(\pi))/2$.

Proof. The theorem is clearly followed from Lemma 3.2 since $b(\pi) - o(\pi) = 0$ if and only if $\pi = \mathcal{I}$, the identity permutation. □

The above lower bound was first obtained in [5] in the case where the operations allowed do not include the third kind of transpositions. Therefore Theorem 3.1 is a generalization of Theorem 3 in [5]. In [5], the authors have proposed an $O(n^2)$ time 2-approximation algorithm *SORT* for sorting a signed permutation using reversals and the first two kinds of transpositions. In fact, we can show that the second kind of transpositions is not really necessary in the algorithm, that is, we can achieve a series of same number of operations each of which is either a reversal or a first kind of transposition. Therefore we have the following theorem:

Theorem 3.2. *There is an* $O(n^2)$ *time* 2*-approximation algorithm for all the three versions of sorting signed permutations.*

4. A better 1.75-approximation algorithm

We consider the third version of sorting signed permutation, that is, sorting by reversals and all three kinds of transpositions. Using the common lower bound established in Section 3, we give a better approximation algorithm in this section.

4.1. Padding into simple permutation

For a signed permutation π, a k-cycle is *long* if $k > 3$ otherwise *short*. A permutation is called *simple* if all the cycles in its breakpoint graph are short. Let $b = (v_b, w_b)$ be a black edge and $g = (w_g, v_g)$ be a gray edge belonging to a cycle $C = \langle \ldots, v_b, w_b, \ldots, w_g, b_g, \ldots \rangle$ in $G(\pi)$. A (g,b)-*split* (this definition and the following *padding* are introduced in [6]) of $G(\pi)$ is a new graph $\hat{G}(\pi)$ obtained from $G(\pi)$ by

- removing edges g and b,
- adding two new vertices v and w,
- adding two new black edges (v_b, v) and (w, w_b),
- adding two new gray edges (w_g, w) and (v, v_g).

Fig. 4 shows a (g,b)-split transforming a cycle C in $G(\pi)$ into two cycles C_1 and C_2 in $\hat{G}(\pi)$. If $G(\pi)$ is a breakpoint graph of a signed permutation π, then every (g,b)-split of $G(\pi)$ corresponds to the breakpoint graph of a signed *generalized* permutation $\hat{\pi}$ such that $\hat{G}(\pi) = G(\hat{\pi})$. Below we define the generalized permutations and describe the

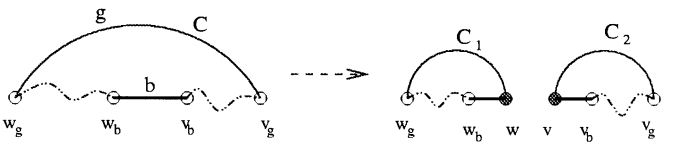


Fig. 4. A (g,b)-split.

padding procedure to find a generalized permutation $\hat{\pi}$ corresponding to a (g,b)-split of $G(\pi)$.

Let $b=(\pi_{i+1},\pi_i)$ be a black edge and $g=(\pi_j,\pi_k)$ be a gray edge belonging to a cycle $C=\langle\ldots,\pi_{i+1},\pi_i,\ldots,\pi_j,\pi_k,\ldots\rangle$ in the breakpoint graph $G(\pi)$. Define $\Delta=\pi_k-\pi_j$ and let $w=\pi_j+\Delta/3$, $v=\pi_k-\Delta/3$. A (g,b)-*padding* of $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$ is a permutation on $n+2$ elements obtained from π by inserting w and v after the ith element of π $(0\leqslant i\leqslant n)$:

$$\hat{\pi}=(\pi_1,\ldots,\pi_i,w,v,\pi_{i+1},\ldots,\pi_n).$$

Note that v and w are both consecutive and adjacent in $\hat{\pi}$, thus implying that if π is a signed permutation then $\hat{\pi}$ is also a signed permutation. The following lemma establishes the correspondence between (g,b)-paddings and (g,b)-splits.

Lemma 4.1. $\hat{G}(\pi)=G(\hat{\pi})$.

It is clear that if g and b are non-adjacent edges of a long cycle C in $G(\pi)$ then the (g,b)-padding breaks C into two smaller cycles in $G(\hat{\pi})$. Therefore paddings may be used to transform an arbitrary permutation π into a simple permutation. If $b(\pi)-o(\pi)=b(\hat{\pi})-o(\hat{\pi})$, then the corresponding padding is called a *safe* padding.

Lemma 4.2. *For any long k-cycle, there exists a safe padding on it such that the long cycle is divided into a 3-cycle and a $(k-2)$-cycle.*

Proof. Let b_1 be the leftmost black edge in this long cycle C. Let b_2 and b_3 denote the black edges that are connected to b_1 via a gray edge. Let g_2 be the gray edge that is connected to b_2 but not b_1. Let g_3 be the gray edge that is connected to b_3 but not b_1. Then both of the (g_2,b_3)-padding and the (g_3,b_2)-padding result in one 3-cycle and one $(k-2)$-cycle (with one operation). Since k is odd if and only if $(k-2)$ is odd, each of the two paddings is safe. This proves the lemma. □

Whenever there is a long cycle, we may applying Lemma 4.2 to find a safe padding to break this long cycle into one 3-cycle and a smaller cycle. This will eventually result in a simple permutation $\hat{\pi}$. A permutation π is *equivalent* to a permutation $\hat{\pi}$ if there exists a series of safe (g,b)-paddings which pad π into $\hat{\pi}$. Therefore we have proved the following theorem.

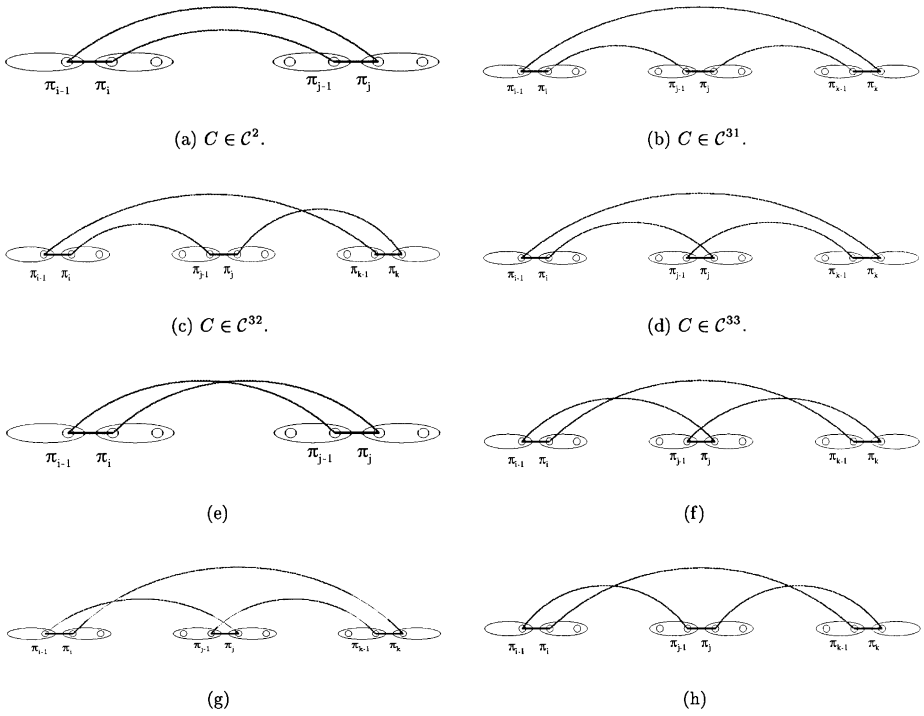


Fig. 5. Configurations of short cycles.

Theorem 4.1. *For every permutation there exists an equivalent simple permutation.*

Let $\hat{\pi}$ be an equivalent simple permutation of π and ρ be an operation acting on two black edges of $\hat{\pi}$. Then ρ can be mimicked on π by ignoring the padded elements. The following lemma has been proved in [6].

Lemma 4.3. *Every sorting of* $\hat{\pi}$, *an equivalent simple permutation of* π, *mimics a* (*genuine*) *sorting of* π *with the same number of operations.*

From the above discussion, we need to consider the elimination of simple permutations only. There are two non-isomorphic configurations for a 2-cycle and six for a 3-cycle, as shown in Fig. 5. We will first describe the elimination of short cycles with configurations shown in Figs. 5(e)–(h), because they are relatively easy to deal with.

If there is a 2-cycle with a configuration as shown in Fig. 5(e), we can eliminate it by a reversal $r(i,j)$. Note that the reversal reduces the number of breakpoints by 2 while keeping the number of odd cycles unchanged. If there is a 3-cycle with a configuration as shown in Fig. 5(f), we can eliminate it by a transposition $t(i,j,k)$. Note that the transposition reduces the number of breakpoints by 3 and the number of odd cycles by 1. If there is a 3-cycle with a configuration as shown in Fig. 5(g), we can eliminate it by a transposition $rt(i,j,k)$. Note that the transposition reduces the number

of breakpoints by 3 and the number of odd cycles by 1. If there is a 3-cycle with a configuration as shown in Fig. 5(h), we can eliminate it by a transposition $rr(i,j,k)$. Note that the transposition reduces the number of breakpoints by 3 and the number of odd cycles by 1. To summarize, if there is a cycle in one of the configurations as shown in Figs. 5(e)–(h), we can use one operation to reduce the parameter $b(\pi)-o(\pi)$ by two. We would like to emphasize that we want to eliminate such cycles this way whenever there is such a cycle in the permutation.

Let $\mathscr{C}^2$ ($\mathscr{C}^{31}, \mathscr{C}^{32}, \mathscr{C}^{33}$) denote the set of cycles with configurations as shown in Figs. 5(a), (b), (c), and (d), respectively) upon reflection. In the following subsections, we will describe the elimination of short cycles in $\mathscr{C}^2 \cup \mathscr{C}^{31} \cup \mathscr{C}^{32} \cup \mathscr{C}^{33}$. These are more involved and require some amortized analysis on the average reduction of the parameter $b(\pi) - o(\pi)$ per operation, and it is necessary to distribute the "weights" of the cycles created in the sorting process in a carefully designed way. In the following arguments, we will define an artificial *weight* of a cycle to represent its contribution to the parameter $b(\pi) - o(\pi)$. At the beginning, any (short) cycle is assigned a weight of 2. During the sorting process, the weight of a 3-cycle never changes. When there are new 2-cycles with a configuration as shown in Fig. 5(a) created during the sorting process, we need to define their weights appropriately. In defining the weights, we will make use of a positive variable x, whose value will be chosen (to be $\frac{1}{7}$) at the end of the discussion.

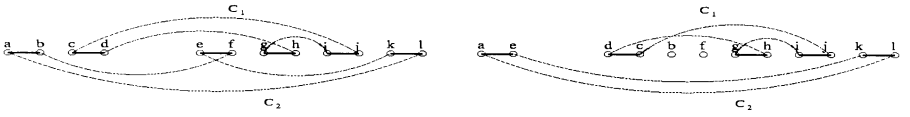
Our goal is to show that every three operations can reduce the parameter $b(\pi) - o(\pi)$ by at least 4. Some consecutive operations have to be grouped together in order to calculate the weight that they reduce. After the execution of each such group of operations, there may be an oriented 2-cycle (as shown in Fig. 5(e)), or a 3-cycle as shown in Figs. 5(f)–(h). If this is the case, we will apply a corresponding operation to eliminate it. By appropriately defining the weights, it is guaranteed that this operation reduces a weight of at least $\frac{4}{3}$.

Let C_1 and C_2 be two 3-cycles, and let $b_1 < b_2 < b_3$ and $b'_1 < b'_2 < b'_3$ be the three black edges in C_1 and C_2, respectively. They are *intersecting* if there are b_i and b'_j such that both $b'_1 < b_i < b'_3$ and $b_1 < b'_j < b_3$ hold. Furthermore, if $b_1 < b'_1 < b_2 < b'_2 < b_3 < b'_3$ or $b'_1 < b_1 < b'_2 < b_2 < b'_3 < b_3$, then they are *interleaving*.

4.2. Eliminating cycles in $\mathscr{C}^{32}$ and $\mathscr{C}^{33}$

Lemma 4.4. *There are two operations which change two intersecting cycles in $\mathscr{C}^{33}$ into a 2-cycle.*

Proof. The key to the proof of the lemma is the following observation: Let $C_1 \in \mathscr{C}^{33}$, and the three black edges in C_1 be $b_1 < b_2 < b_3$. Clearly, if b_1 can be reversed such that its relative position is unchanged with respect to b_2 and b_3, or the segment containing b_2 and b_3 (but not b_1) can be reversed (this may be regarded equivalent to the former case if we circularize the linear permutation), then C_1 is changed into a cycle as shown in Fig. 5(g) or a cycle as shown in Fig. 5(h), and can be subsequently eliminated by



(a) two intersecting cycles in $\mathcal{C}^{33}$ (b) after a reversal $r(b,f)$

Fig. 6. Proof of Lemma 4.4.

a suitable transposition rt or by a suitable transposition rr. For any two intersecting 3-cycles in $\mathcal{C}^{33}$, we can find a reversal r which simultaneously changes one of the 3-cycles into a 2-cycle and the other into a 3-cycle as shown in either Fig. 5(g) or Fig. 5(h).

For example, let us see a configuration as shown in Fig. 6(a): where $b'_1 < b'_2 < b'_3$ are the three black edges in cycle C_2, with $b'_1 < b_1 < b'_2 < b_2 < b_3 < b'_3$ ($b_1 = (a,b)$, $b'_1 = (c,d)$, $b_2 = (e,f)$, $b'_2 = (g,h)$, $b'_3 = (i,j)$, and $b_3 = (k,l)$). Applying a reversal on the black edges b'_1 and b'_2 will reverse edge b_1 and change C_2 into a 2-cycle, resulting a configuration as shown in Fig. 6(b). Denote these two cycles still by C_1 and C_2. We may now apply a suitable transposition rt to eliminate cycle C_1, which acts on the three black edges of C_1. Therefore, after two operations, there is only a 2-cycle left. The other cases can be similarly discussed. □

The 2-cycle left after executing the two operations in Lemma 4.4 is either oriented or unoriented. In the former case, we further apply a suitable reversal to eliminate it. In the latter case, we amortize its weight to be $2-2x$. It follows that

Corollary 4.1. *Two intersecting cycles in $\mathcal{C}^{33}$ can be either changed by two operations into a 2-cycle in $\mathcal{C}^2$ with a weight of $2-2x$, or eliminated by three operations. Therefore each of the operations reduces a weight of at least* $\min\{1+x, \frac{4}{3}\}$, *on the average.*

Using Lemma 4.4, whenever there is a pair of two intersecting 3-cycles in $\mathcal{C}^{33}$, we perform the corresponding operations to either eliminate them or change them into a 2-cycle with a weight of $2-2x$. We remark that these two or three operations should be grouped together in our analysis. If a cycle as shown in Figs. 5(e)–(h) occurs between two groups of operations, we will apply a suitable operation to eliminate it immediately. Notice that till now each operation reduces a weight of at least $\min\{1+x, \frac{4}{3}, 2-2x\}$, on the average.

Similar to the observation made in the proof of Lemma 4.4, we notice that for a 3-cycle in $\mathcal{C}^{32}$ as shown in Fig. 5(c) (the three black edges are $b_1 < b_2 < b_3$): if edge b_1 can be reversed such that its relative position is unchanged with respect to edges b_2 and b_3, or if edge b_2 can be reversed such that its relative position is unchanged with respect to edges b_1 and b_3, or if the segment containing edges b_2 and b_3 (but not b_1) can be reversed, then the resultant 3-cycle can be eliminated by one suitable transposition. By enumerating all possible configurations, Lemmas 4.5–4.7 can be similarly proved.

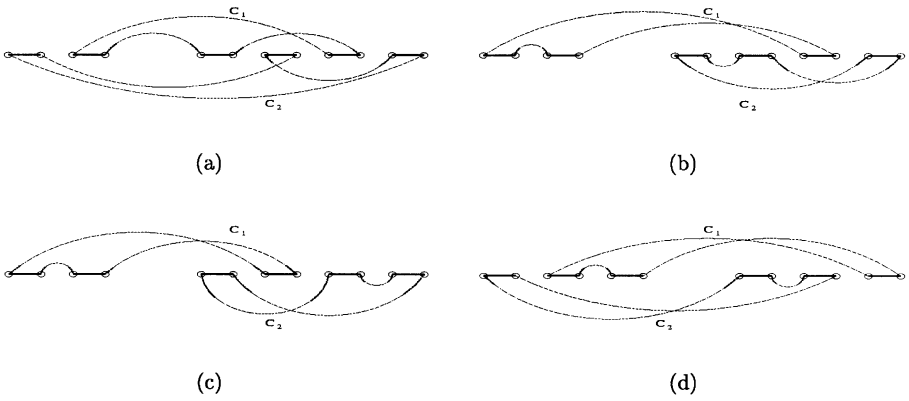


Fig. 7. Configurations that cannot be eliminated by operations in Lemmas 4.4–4.7.

Lemma 4.5. *Let C_1 be a cycle in $\mathscr{C}^{32}$ and C_2 a cycle in $\mathscr{C}^{33}$, if C_1 and C_2 are intersecting, then they can be*

1. *either changed by two operations into a 2-cycle in $\mathscr{C}^2$ with a weight of $2-2x$;*
2. *or eliminated by three operations;*

except that they from a configuration as shown in Fig. 7(a). Therefore, each of the operations reduces a weight of at least $\min\{1+x, \frac{4}{3}\}$, on the average.

Lemma 4.6. *Two intersecting cycles in $\mathscr{C}^{32}$ can be*

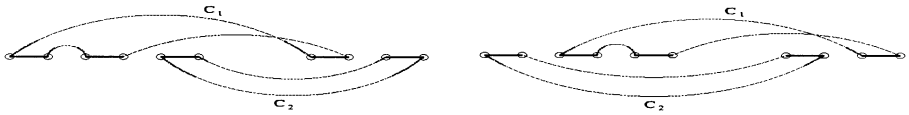
1. *either changed by two operations into a 2-cycle in $\mathscr{C}^2$ with a weight of $2-2x$;*
2. *or eliminated by three operations;*

except that they from a configuration as shown in Figs. 7(b)–(d). Therefore, each of the operations reduces a weight of at least $\min\{1+x, \frac{4}{3}\}$, on the average.

Lemma 4.7. *Let C_1 be a 3-cycle, and C_2 a 3-cycle in $\mathscr{C}^{31}$. If C_1 and C_2 are interleaving, then they can be eliminated by three operations. Therefore, each of the operations reduces a weight of $\frac{4}{3}$, on the average.*

Since every 3-cycle has a weight of 2 and every 2-cycle occurred so far has a weight at least $2-2x$, every operation performed reduces a weight of at least $\min\{1+x, \frac{4}{3}, 2-2x\}$. We may extend the definition of intersecting of two 3-cycles to a 3-cycle and a 2-cycle. That is, let $b_1<b_2<b_3$ be the three black edges in a 3-cycle C_1 and $b'_1<b'_2$ the two in a 2-cycle C_2 (in $\mathscr{C}^2$). C_1 and C_2 are *intersecting* if there is b_i such that $b'_1<b_i<b'_2$ and there is b'_j such that $b_1<b'_j<b_3$.

Lemma 4.8. *Let C_1 be a cycle in $\mathscr{C}^{32}\cup\mathscr{C}^{33}$, and C_2 a cycle in $\mathscr{C}^2$. Suppose C_1 and C_2 are intersecting, then they can be eliminated by three operations except that they form a configuration as shown in Figs. 8(a), (b). Therefore, each of the operations reduces a weight of at least $(4-2x)/3$, on the average.*



(a) (b)

Fig. 8. The configurations that cannot be eliminated by operations in Lemma 4.8.

Proof. The key to the proof is still the observations made in the proof of Lemma 4.4. The proof is done again by enumerating all possible configurations. Note that up to now, every 2-cycle in $\mathscr{C}^2$ in the graph has a weight of at least $2-2x$. It follows that the total weight of these two cycles is at least $4-2x$. Therefore, each of the operations reduces the parameter by at least $(4-2x)/3$, on the average. □

We will apply one of Lemmas 4.4–4.8, or a suitable operation that eliminates one cycle, until none of the lemmas applies. As a result, we get a graph in which there is no configuration as shown in Figs. 5(e)–(h), nor as stated in Lemmas 4.4–4.8, except the ones shown in Figs. 7 and 8. We remark that till now every operation performed so far reduces a weight of at least $\min\{1+x, 2-2x, (4-2x)/3\}$. We further extend the definition of intersecting (interleaving) to two 2-cycles. Clearly, two 2-cycles are intersecting if and only if they are interleaving. From now on we prohibit any reversals. We pad each remaining cycles in $\mathscr{C}^{32} \cup \mathscr{C}^{33}$ into two 2-cycles as follows: Suppose $b_1 < b_2 < b_3$ are the three black edges in cycle $C \in \mathscr{C}^{32}$ as shown in Fig. 5(c), the padding is performed on black edge b_2 (and the unique gray edge to break the 3-cycle into two 2-cycles). Suppose $b_1 < b_2 < b_3$ are the three black edges in cycle $C \in \mathscr{C}^{33}$ as shown in Fig. 5(d), the padding is performed on black edge either b_1 or b_3 (arbitrarily). Note that this will pad C into two non-intersecting 2-cycles, of which exactly one is in $\mathscr{C}^2$, denoted by C_1. We amortize the weight of C_1 to be $(1-x)$ while the other 2-cycle, which is oriented, is assigned a weight of $(1+x)$.

Claim 4.1. *After padding all the remaining* 3*-cycles in* $\mathscr{C}^{32} \cup \mathscr{C}^{33}$ *into* 2*-cycles, there is no such case that a* 2*-cycle* C *in* $\mathscr{C}^2$ *with weight* $(1-x)$ *intersects another* 2*-cycle.*

Proof. Suppose to the contrary that C intersects another 2-cycle C'. From Lemma 4.8 we derive that C' cannot be a 2-cycle with a weight of 2 or $2-2x$. This is true because otherwise we may apply Lemma 4.8 to the two cycles C' and the 3-cycle, say C^*, from which cycle C is padded. It is also impossible, for the same reason, by Lemmas 4.5 and 4.6 that C' has a weight of $1-x$ or $1+x$. However, any 2-cycle existing in the graph has a weight of 2, $2-2x$, $1+x$ or $1-x$. In other words, such a cycle C' does not exist. □

Since there should be a gray edge connecting an element lying between the two black edges of a 2-cycle in $\mathscr{C}^2$ and an element outside, any such a 2-cycle must intersect with some other cycle. After the padding, every 3-cycle is in $\mathscr{C}^{31}$. Thus we have

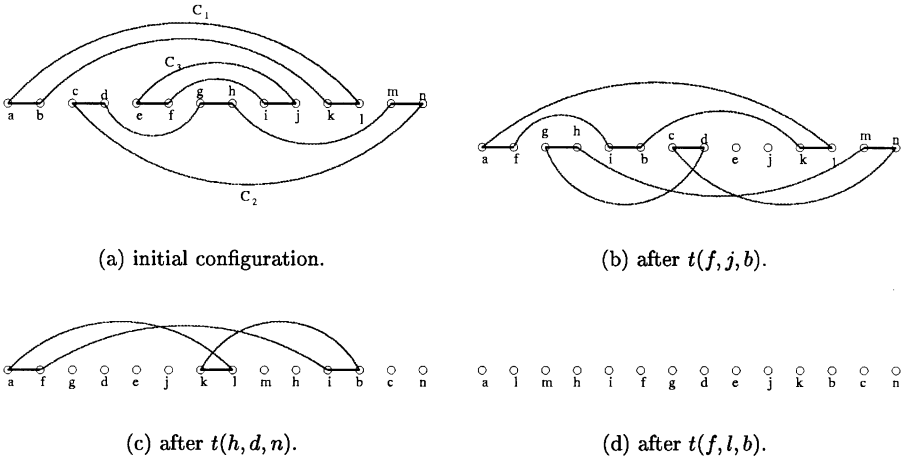


Fig. 9. Proof of Lemma 4.9, one case.

Corollary 4.2. *A 2-cycle C in $\mathscr{C}^2$ with weight $(1-x)$ intersects a 3-cycle in $\mathscr{C}^{31}$.*

4.3. Eliminating cycles in $\mathscr{C}^{31}$

For a 3-cycle in $\mathscr{C}^{31}$, let $b_1 < b_2 < b_3$ denote the three black edges in it. Again, it is clear that there should be a gray edge connecting an element lying between b_1 and b_2 (b_2 and b_3) and an element outside. The segment between b_1 and b_2 (b_2 and b_3) is called a *gap* of the 3-cycle. The gray edge belongs to another cycle, which is said to *intersect* the 3-cycle in the gap.

Lemma 4.9. *Let C_1 be a 2-cycle in $\mathscr{C}^2$ with weight $(1-x)$, and it intersects C_2, a 3-cycle in $\mathscr{C}^{31}$, in a gap of C_2. If there is another cycle C_3 in $\mathscr{C}^2$ intersecting C_2 in the other gap, then these three cycles can be eliminated by three first kind of transpositions t's. Therefore, each of the three transpositions reduces a weight of at least $(4-2x)/3$, on the average.*

Proof. First we note that C_1 and C_3 cannot intersect each other by Claim 4.1. There are five configurations that need to be considered, according to how cycles C_1 and C_3 intersect cycle C_2. We just discuss one of them here, the other four can be similarly done. Assume that these three cycles intersect as shown in Fig. 9(a). Applying transposition $t(f,j,b)$, we get a configuration as shown in Fig. 9(b). Applying transposition $t(h,d,n)$, we get a configuration as shown in Fig. 9(c). Then applying transposition $t(f,l,b)$, we get a configuration as shown in Fig. 9(d), in which the seven black edges are all eliminated.

Since C_3 has a weight of at least $(1-x)$ and C_2 has a weight of 2, these three first kind of transpositions reduce a weight of at least $(4-2x)$. □

Note that a first kind of transposition never makes an unoriented gray edge oriented. We have the following corollary.

Corollary 4.3. *Eliminating the three cycles in Lemma* 4.9 *does not change the configuration of any other* 2*-cycles. And if a* 3*-cycle in* $\mathscr{C}^{31}$ *is changed into a configuration as shown in Fig.* 5(f), *then perform another first kind of transposition to eliminate it. This again does not change the configuration of any other* 2*-cycles.*

Using a similar logic, and enumerating all possible configurations, we can prove the following lemma.

Lemma 4.10. *Let* C_1 *be a* 2*-cycle in* $\mathscr{C}^2$ *with weight* $(1-x)$ *which intersects* C_2, *a* 3*-cycle in* $\mathscr{C}^{31}$, *in a gap of* C_2. *If there is another* 3*-cycle* C_3 (*in* $\mathscr{C}^{31}$) *intersecting* C_2 *in the other gap of* C_2, *we pad* C_3 *into two* 2*-cycles with the one* (*call it* C_{31}) *intersecting* C_2 *in that gap having a weight* $(3-x)/5$ (*and the other* C_{32} *having a weight* $(7+x)/5$). *The three cycles* C_1, C_2 *and* C_{31} *can be eliminated by three first kind of transpositions, and thus each of the transpositions reduces a weight of at least* $(6-2x)/5$, *on the average.*

Corollary 4.4. *The elimination of the three cycles in Lemma* 4.10 *does not change the configuration of any other* 2*-cycle. If a* 3*-cycle in* $\mathscr{C}^{31}$ *is changed into a configuration as shown in Fig.* 5(f), *we may perform another first kind of transposition to eliminate it. This transposition reduces a weight of* 2, *and does not change the configuration of any other* 2*-cycle. Moreover, if the cycle* C_{32} *interleaves another* 2*-cycle in* $\mathscr{C}^2$ *with weight* $(1-x)$, *then they can be eliminated by two first kind of transpositions, which reduce a total weight of* $(12-4x)/5$.

Note that a 2-cycle in $\mathscr{C}^2$ with weight $(3-x)/5$ is eliminated right after it is generated. Therefore there is no such cycle remaining at the time when starting a group of consecutive operations. After applying Lemmas 4.9 and 4.10 to the permutation, if there still exists a 2-cycle in $\mathscr{C}^2$ with weight $(1-x)$, there should be an oriented 2-cycle intersecting the other gap of the concerned 3-cycle. This case can be handled as in the following lemma.

Lemma 4.11. *Let* C_1 *be a* 2*-cycle in* $\mathscr{C}^2$ *with weight* $(1-x)$ *which intersects* C_2, *a* 3*-cycle in* $\mathscr{C}^{31}$, *in one of the gaps of* C_2. *If there is no cycle in* $\mathscr{C}^2 \cup \mathscr{C}^{31}$ *intersecting* C_2 *in the other gap, there should be an oriented* 2*-cycle* C_3 *intersecting* C_2 *in the other gap. Moreover, the three cycles* C_1, C_2 *and* C_3 *can be eliminated by the two first kind of transpositions together with a second kind of transposition. These three operations reduce a weight of at least* 4 *and they never transform an unoriented* 2*-cycle into an oriented one.*

Proof. The proof is similar to that of Lemma 4.9, noticing additionally that only the black edges in the second gap of C_2 are reversed, none of which belongs to any intersecting cycle in $\mathscr{C}^2 \cup \mathscr{C}^{31}$. □

Apply any one of Lemmas 4.9, 4.10 and 4.11, whenever applicable, to eliminate at least one cycle in $\mathscr{C}^{31}$, and one cycle in $\mathscr{C}^{2}$ with a weight $(1-x)$. After that, we see that there is no $(1-x)$-weight 2-cycle in $\mathscr{C}^{2}$ any more. But there may be some cycles in $\mathscr{C}^{31}$ left. Let C_1 be such a 3-cycle. If there is another 3-cycle C_2 in $\mathscr{C}^{31}$ intersecting C_1 (only in one gap by Lemma 4.7), then pad C_2 into C_{21} which intersects C_1 and has a weight $(4-3x)/5$ and C_{22} which has a weight $(6+3x)/5$.

Lemma 4.12. *Let C_1 be a 3-cycle in $\mathscr{C}^{31}$. Let C_2 and C_3 be two 2-cycles intersecting C_1 in different gaps. Then these three cycles can be eliminated by three operations, which reduce a total weight of at least $(18-6x)/5$.*

Proof. The proof is done similarly by noticing that at this time any unoriented 2-cycle has a weight at least $(4-3x)/5$ and the weight of a 3-cycle is always 2. □

4.4. Eliminating remaining cycles

Note that by priority, the operations performed in Lemmas 4.9–4.12 never transform an unoriented 2-cycle with weight less than $\min\{1+x,\ 2-2x,\ (7+x)/5,\ (6+3x)/5\}$ into an oriented one; and they never transform a 3-cycle in $\mathscr{C}^{31}$ into one in $\mathscr{C}^{32}\cup\mathscr{C}^{33}$. Therefore, after all 3-cycles in $\mathscr{C}^{31}$ are eliminated, there are only 2-cycles with weights no less than $\min\{1+x,\ 2-2x,\ (7+x)/5,\ (6+3x)/5\}$ remaining. At this time, we cancel the prohibit of eliminating oriented 2-cycles by a reversal, that is, whenever there is an oriented 2-cycle, perform a reversal to eliminate it. Obviously, this reversal reduces a weight of at least $\min\{1+x,\ 2-2x,\ (7+x)/5,\ (6+3x)/5\}$.

When there is no oriented 2-cycle left, using the technique developed in [5, Lemma 7] by performing two first kind of transpositions to eliminate a pair of interleaving unoriented 2-cycles. Each of these two transpositions reduces a weight of at least $\min\{1+x,\ 2-2x,\ (7+x)/5,\ (6+3x)/5\}$.

4.5. The 1.75-approximation algorithm

Note that a series of operations performed on the equivalent simple permutation mimics a series of operations performed on the initial permutation. Therefore, we have an algorithm which sorts any permutation into identity. The sketch of the algorithm goes as follows: Given any signed permutation, transform it into an unsigned image permutation. After constructing the break point graph, pad any long cycle into two shorter cycles according to Lemma 4.2. Do the cleaning up job, that is, if there is any cycle as shown in Figs. 5(e)–(h), perform a suitable operation to eliminate it. Then whenever one of Lemmas 4.4–4.8 is applicable, apply suitable (two or three) operations to either eliminate those involved cycles or change them into one $(2-2x)$-weight unoriented 2-cycle. Do the cleaning up job again and repeat this process until impossible. At this time, pad every remaining 3-cycle in $\mathscr{C}^{32}\cup\mathscr{C}^{33}$ appropriately into an unoriented 2-cycle assigned with a weight $(1-x)$ and an oriented 2-cycle assigned with a weight $(1+x)$. But reversals are prohibited temporarily. Examine if Lemma 4.9,

and then Lemma 4.10, and then Lemma 4.11, and then Lemma 4.12, is applicable. If so, some suitable operations (and possible additional padding on a 3-cycle in $\mathscr{C}^{33}$) are performed to eliminate involved cycles. These operations are guaranteed not to transform an unoriented 2-cycle into an oriented one. Repeat again until impossible. We get a graph in which all cycles are 2-cycles, each of which has a weight at least $\min\{1+x,\ 2-2x,\ (7+x)/5,\ (6+3x)/5\}$. Now whenever there is an oriented cycle, apply a reversal to eliminate it. The resultant graph consists of unoriented 2-cycles only, for which we may apply Lemma 7 stated in [5] to eliminate all of them. The algorithm is summarized in Fig. 10. Note that the average reduction of the operations performed in the algorithm is at least $\min\{1+x,\ (4-2x)/3,\ (6-2x)/5,\ 2-2x\}$, which is $\frac{8}{7}$ when $x=\frac{1}{7}$. Now using the lower bound on the number of operations needed in Theorem 3.1 and the same logic as used in the proof of Theorem 3.2, we have the following theorem.

Algorithm BSORT(π)

Input: a signed permutation π.
Output: a series of operations transforming π into $\mathscr{I}$.

Begin
1. Construct $G(\pi)$;
2. **While** (there is a long cycle in $G(\pi)$) **Do**
   pad it into two cycles by Lemma 4.2;
3. **While** (there is a black edge in $G(\pi)$) **Do** {
3.1 clean_up( );
3.2 **While** (one of Lemmas 4.4–4.8 applicable) **Do** {
3.2.1 do it;
3.2.2 clean_up( );
   }
3.3 padding remaining cycles in $\mathscr{C}^{32} \cup \mathscr{C}^{33}$;
3.4 **While** (one of Lemmas 4.9–4.12 applicable) **Do**
   do it according to priority;
3.5 clean_up( );
3.6 Apply [5, Lemma 7] to each pair of interleaving unoriented 2-cycles.
  }
End

clean_up( ):
While (there is a cycle as shown in Figs. 5(e)–5(h)) **Do**
  eliminate it;

Fig. 10. The 1.75-approximation algorithm.

Theorem 4.2. *Algorithm* BSORT *is an* $O(n^2)$ *time* 1.75*-approximation algorithm for the third version of sorting signed permutations, i.e., using reversals and all three kinds of transpositions.*

Proof. Note that the total weight of the break point graph is $b(\pi) - o(\pi)$. Since every operation reduces a weight of at least $\frac{8}{7}$ on the average, the total number of operations required by BSORT to sort the given permutation is no more than $(b(\pi) - o(\pi))\frac{7}{8}$ which is at most $\frac{7}{4}d(\pi)$ by Theorem 3.1. Therefore BSORT has a performance ratio of at most 1.75.

Since the padding is performed $O(n)$ times and each padding takes $O(n)$ time. The running time of BSORT is $O(n^2)$. □

5. Concluding remarks

In this paper, we studied sorting of a signed permutation by reversals and transpositions, a problem which adequately models genome rearrangement. Three variants of transpositions, and thus three corresponding rearrangement problems are considered. We establish a common lower bound on the number of operations needed, and present a unified 2-approximation algorithm for all three problems. By exploiting more structure properties of the possible configurations of short cycles, we presented a better 1.75-approximation for the third problem. Better approximations for all the three problems, as well as their computational complexities, are left open.

Acknowledgements

We would like to thank an anonymous referee whose comments greatly improved the presentation of the paper.

References

[1] V. Bafna, P.A. Pevzner, Sorting by reversals: genome rearrangements in plant organelles and evolutionary history of X chromosome, Mol. Biol. Evol. 12 (1995) 239–246.
[2] V. Bafna, P.A. Pevzner, Genome rearrangements and sorting by reversals, SIAM J. Comput. 25 (2) (1996) 272–289.
[3] V. Bafna, P.A. Pevzner, Sorting by transpositions, SIAM J. Discrete Math. 11 (2) (1998) 224–240.
[4] A. Caprara, Sorting by reversals is difficult, in: Proc. 1st Annu. Internat. Conf. on Computational Molecular Biology, 1997, pp. 75–83.
[5] Q.-P. Gu, S. Peng, H. Sudborough, A 2-approximation algorithm for genome rearrangements by reversals and transpositions, Theoret. Comput. Sci. 210 (1999) 327–339.
[6] S. Hannenhalli, P. Pevzner, Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals, J. ACM 46 (1) (1999) 1–27.
[7] S. Kececioglu, D. Sankoff, Exact and approximation algorithms for the inversion distance between two permutations, Algorithmica 13 (1995) 180–210.